Devoir Surveillé 2 : Correction

1 Théorèmes de Morgan

Question 1. Énoncer les deux théorèmes de Morgan

not(A or B) = not A and not B
not(A and B) = not A or not B

Question 2. Démontrer un des deux théorèmes de Morgan, après avoir précisé celui que vous choisissez de démontrer.

voir cours ; dresser une table de vérité.

2 Expressions booléennes

Question 3. Écrire l'opérateur booléen "xor" de deux manières différentes. On rappelle le tableau de vérité de xor :

A	B	A xor B
True	True	False
True	False	True
False	True	True
False	False	False

Il existe bien sûr des variantes possibles à la réponse, néanmoins les deux formes normales (vues en TD !) sont :

A xor B = (A or B) and not(A and B)
A xor B = (A and not B) or (not A and B)

Question 4. Dans une pièce, il y a trois interrupteurs A, B, C. La pièce est allumée quand un nombre impair d'interrupteurs est en position `True` (qui équivaut à un interrupteur fermé) et elle est éteinte sinon.
Écrire, en justifiant, l'expression booléenne L qui donne la valeur `True` quand la pièce est allumée et `False` quand elle est éteinte en fonction de A, B et C.

On se convaincra d'autant plus facilement que L=A xor B xor C que l'on reverra l'exercice 2 du cours sur les boucles.
À défaut de faire le lien avec la question précédente, on peut lister les possibilités :
L = (A and not B and not C) or (not A and B and not C) or (not A and not B and C) or (A and B and C)

3 Algorithme d'Euclide

Question 5. Écrire une fonction `Euclide(a,b)` qui prend en arguments deux entiers positifs `a` et `b` et qui détermine leur PGCD calculé à l'aide de l'algorithme d'Euclide par division successives.

```
def Euclide(a,b):
    while b>0 : # On suppose que a et b sont >0
        r=a%b
        a=b
        b=r
    return a
```

Question 6. Écrire la ligne de commande, saisie dans la console, qui affiche le PGCD de $10^{10}+2$ et de $10^{20}+2$.

`print(Euclide(10**20+2,10**10+2))`.

Il est à noter que puisque la commande est saisie dans la console, `print` n'est pas obligatoire, l'interpréteur Python affichant un résultat dès lors qu'il le peut pour les instructions saisies dans la console.

4 Variables globales et locales

Question 7. Donner la définition d'une variable globale et d'une variable locale.

Voir cours. Une variable globale est définie en dehors d'une fonction. Sa portée est tout le programme (toute la session en fait). Elle est accessible depuis une fonction, mais n'y est pas (a priori) modifiable. Une variable locale est définie dans une fonction. Sa portée est la fonction. En dehors de cette fonction, cette variable n'existe pas.

5 Tour de puissances

Question 8. Écrire une fonction `tour(n,k)` prenant en arguments deux entiers positifs `n` et `k` et qui renvoie la n-ième puissance itérée de k, autrement dit $k^{k^{\cdot^{\cdot^{\cdot^{k}}}}}$ formé de n exemplaires de k.

```
def tour(n,k):
    p=1
    for i in range(n):
        p=k**p
    return p
```

6 Intégration numérique

On rappelle la formule de calcul approché d'une intégrale par la méthode des rectangles à gauche :

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{i=0}^{i=n-1} f\left(a+i\frac{b-a}{n}\right)$$

Question 9. Écrire une fonction `rectangles(f,a,b,n)` qui prend en arguments une fonction f, des flottants a et b et un entier n et qui renvoie le résultat du calcul approché de l'intégrale de a à b de la fonction f en n pas telle que donnée par la formule écrit plus haut.

```
def rectangles(f,a,b,n):
    s=0
    pas=(b-a)/n # Pas obligatoire mais simplifie le programme tant en lecture qu'en nombre d'opérations
    for i in range(n): # On s'arrête à n-1 à l'aide de range(n)
        s=s+f(a+i*pas)
    return s*pas
```

Question 10. Écrire une suite de commandes (on pourra éventuellement définir une fonction) qui calcule en 10^5 pas par la méthode des rectangles à gauche l'intégrale suivante :

$$\int_0^1 \frac{x}{1+x^3}dx$$

Une première solution consiste à définir une fonction et à la passer à la fonction `rectangles`. Ce qui donne :

```
def f(x):
    return x/(1+x**3)

print(rectangles(f,0,1,100000))
```

Il est également possible d'utiliser une fonction anonyme, à l'aide du mot clé `lambda` et d'écrire :

```
print(rectangles(lambda x:x/(1+x**3),0,1,100000))
```

7 Contrôle de rendu de monnaie

On s'intéresse dans cet exercice au fonctionnement d'une machine qui rend de la monnaie, selon un algorithme pour nous inconnu.

Cette machine travaille avec deux listes de nombres : une liste `denomination` et une liste `rendu`.

La liste `denomination` contient une liste d'entiers ou de flottants, rangée par ordre strictement décroissant, et correspondant aux billets et/ou pièces dont dispose la machine. Par exemple, dans le système monétaire européen, si la machine peut rendre des billets de 5, 10 et 20 euros, et toutes les pièces possibles, la liste serait :

```
denomination=[20,10,5,2,1,0.5,0.2,0.1,0.05,0.02,0.01]
```

La liste `rendu` est de la même longueur que la liste `denomination` et contient des entiers positifs. Ces entiers correspondent aux nombres de pièces et/ou billets associés à la liste `denomination` que la machine doit rendre. Par exemple, avec la liste `denomination` précédente, la liste :

`rendu=[0,1,0,0,1,1,0,0,0,0,0]`
correspondrait à un rendu d'un billet de 10 euros, un pièce de 1 euro et un pièce de 50 centimes, pour un montant à rendre de 11,50 euros.

Question 11. Écrire une fonction `testrendu(L,D)` qui prend en argument une liste `L` et qui vérifie si cette liste est susceptible d'être une liste de rendu de monnaie (si c'est une liste d'entiers positifs) associée à la liste `D` qui est une liste de dénomination supposée valide. Cette fonction renverra un booléen.

De manière générale dans cet exercice, nous allons utiliser le fait que `return` interrompt l'exécution d'une fonction.
Dans cette question spécifiquement, après avoir testé que `L` et `D` ont la même longueur, nous testerons les éléments de `L`, en utilisant un parcours par éléments :

```
def testrendu(L,D):
    if len(L)!=len(D): # différent de s'écrit != et non =!
        return False
    for x in D:
        if type(x)!=int or x<=0:
            return False
    return True
```

Question 12. Écrire une fonction `testdenomination(D)` qui prend en argument une liste `D` et qui vérifie si cette liste est susceptible d'être une liste de dénominations. Une liste de dénomination valide est une liste d'entiers ou flottants strictement décroissante et dont tous les termes sont strictement positifs.. Cette fonction renverra un booléen.

Un parcours par indices, jusqu'à l'avant dernier élément de `D`, donc en utilisant `range(len(D)-1)` est ici le plus simple. Le premier test aurait pu être incorporé dans la boucle, au prix de davantage de tests nécessaires :

```
def testdenomination(D):
    if type(D[0])!=int and type(D[0])!=float:
        return False
    for i in range(len(D)-1):
        if type(D[i+1])!=int and type(D[i+1])!=float or D[i]<=D[i+1]: #and est prioritaire sur or
            return False
    return True
```

Question 13. Écrire une fonction `monnaie(L,D)` qui prend en arguments deux listes `L` et `D`, qui sont ici supposées être des listes valides de rendu et de dénomination, et qui renvoie le montant total rendu par la machine.

Il s'agit de faire la somme des `L[i]*D[i]` ici. Un parcours par indices est pratiquement indispensable ici :

```
def monnaie(L,D):
    s=0
    for i in range(len(D)): # dans cette question on a par hypothèse len(L)=len(D)
        s=s+L[i]*D[i]
    return s
```

Question 14. Écrire une fonction `testretour(L,D,x)` qui prend en argument deux listes `L` et `D`, représentant `a priori` des listes de rendu et de dénomination, et un nombre `x` et qui renvoie le booléen :
- `False` si les listes `L` et `D` ne sont pas des listes de rendu et de dénomination valides ou si le montant à rendre associé aux listes `L` et `D` n'est pas égal à `x` ;
- `True` si les listes `L` et `D` sont valides et que le montant associé à ces listes et est égal à `x`

On se servira des fonctions précédemment codées dans cet exercice, même si on n'a pas su les coder.

Le cours sur le codage des flottants ne doit pas être enterré au fond du jardin des connaissances informatiques. On rappelle que pour tester l'égalité de deux flottants, il est hors de question d'écrire un simple $==$; la comparaison de flottants se fait toujours à un ε près, les erreurs d'arrondis ne permettant pas de garantir que deux flottants en principe égaux seront effectivement égaux au bit près, particulièrement après des opérations. Il suffit de se rappeler que $0.1+0.1+0.1\neq 0.3$ ...
Il est raisonnable ici de se dire que les systèmes de dénominations employées à travers le monde utilisent des centièmes d'unités. Un ε de l'ordre de 10^{-6} paraît justifié (mais on pourrait utiliser une marge d'erreur plus faible).

```
def testretour(L,D,x):
    if testdenomination(D) and testrendu(L,D): #Ces deux fonctions renvoient des booléens!
        if abs(x-monnaie(L,D))<10**-6:
            return True
    return False
```

Question 15. Avec la liste dénomination donnée en exemple au début de l'exercice, l'algorithme, pour le montant de 9 centimes crée la liste de rendu suivante :

`[0,0,0,0,0,0,0,0,1,1,1]`

Expliquer comment cette erreur a pu se produire, et ce que vous pourriez faire pour que cette erreur ne se produise pas (8 lignes maximum)

Les calculs effectués avec des flottants sont par nature approximatifs. Seuls les décimaux qui sont des sommes de puissances de 2 négatives peuvent donner lieu à des calculs exacts. Ce n'est pas le cas de nombres comme 0,1 qui si ils s'écrivent de manière exacte en base 10 ne peuvent s'écrire de manière exacte en base 2, quelque soit le nombre de bits employés. Rappelons que lors de calculs flottants, $0.1+0.1=0.2$ mais $0.1+0.1+0.1\neq 0.3$!

Ici, nous pouvons conjecturer que l'algorithme (dont nous ne disposons pas) est victime de ces erreurs d'approximations. Pour être certain que les calculs effectués donnent des résultats exacts, ce qui est absolument indispensable dans le cadre d'une machine à rendre la monnaie industrielle, nous travaillerons avec des entiers. Dans le cadre d'un système de dénominations où les unités seraient divisées en centièmes, il s'agirait de multiplier toutes les valeurs par 100 et de convertir l'ensemble des dénominations (et le prix à rendre) en entiers.

8 Boucles imbriquées

Question 16. Écrire une fonction `somme1` d'argument `n` qui retourne la valeur de la somme suivante :

$$\sum_{i=0}^{i=n}\sum_{j=0}^{j=n}\frac{1}{i^2+j^2+1}$$

Le titre de l'exercice est une indication en soi ... On prendra garde à bien effectuer les sommations pour des indices allant jusqu'à n inclus, en utilisant `range(n+1)` :

```
def somme1(n):
    s=0
    for i in range(n+1):
        for j in range(n+1):
            s=s+1/(i**2+j**2+1)
    return s
```

Question 17. Quelle est la complexité de l'algorithme précédemment écrit ? On utilisera la notation $\mathcal{O}$

L'algorithme précédemment écrit a une complexité en $\mathcal{O}(n^2)$

Question 18. Écrire une fonction `somme2` d'argument `n` qui retourne la valeur de la somme suivante :

$$\sum_{1\leqslant i\leqslant j\leqslant n}(i^2-\sqrt{j})$$

Deux implémentations sont ici possibles, suivant la réécriture choisie pour la double somme.
Celle-ci peut être vue comme :

$$\sum_{j=1}^{n}\sum_{i=1}^{j}(i^2-\sqrt{j})$$

ce qui donne l'implémentation suivante :

```
def somme2(n):
    s=0
    for j in range(n+1):
        for i in range(j+1):
            s=s+i**2-j**0.5
    return s
```

Il est également possible de voir la somme comme :

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(i^2-\sqrt{j})$$

et donne l'implémentation suivante :

```
def somme2(n):
    s=0
    for i in range(n+1):
        for j in range(i,n+1):
            s=s+i**2-j**0.5
    return s
```

C'est à chacun de se faire son opinion sur ce qui paraît plus naturel. À titre personnel, je trouve la première implémentation plus intuitive. Comme une majorité de monde. Je suppose que c'est une question de latéralité ; comme je confonds toujours la droite et la gauche en voiture, mon avis personnel est peut-être à prendre avec la plus grande circonspection.

Question 19. On se place dans la suite de cet exercice dans un repère orthonormal de l'espace usuel.

On rappelle qu'une sphère de rayon R et centrée en (a, b, c) a pour équation $(x-a)^2+(y-b)^2+(z-c)^2 = R^2$

Écrire une fonction `pointsentiers(a,b,c,R)` qui prend en argument des entiers a,b, et c correspondant au centre d'une sphère, un entier R correspondant à son rayon et qui affiche à l'écran l'ensemble des points (x,y,z) à coordonnées **entières** (x, y et z doivent tous les trois être entiers) et qui appartiennent à cette sphère.

Nous allons faire parcourir à x,y et z toutes les combinaisons possibles de triplets d'entiers entre 0 et R. Lorsqu'un point conviendra, nous afficherons les huit points appartenant à la sphère obtenus par symétrie.

```
def pointsentiers(a,b,c,R):
    for i in range(R+1):
        for j in range(R+1):
            for k in range(R+1):
                if i**2+j**2+k**2==R**2:
                    print(a-i,b-j,c-j)
                    print(a+i,b-j,c-j)
                    print(a-i,b+j,c-j)
                    print(a+i,b+j,c-j)
                    print(a-i,b-j,c+j)
                    print(a+i,b-j,c+j)
                    print(a-i,b+j,c+j)
                    print(a+i,b+j,c+j)
```

Question 20. Quelle est la complexité de l'algorithme précédemment écrit ? On utilisera la notation $\mathcal{O}$

L'algorithme précédemment écrit a une complexité en $\mathcal{O}(n^3)$

Question 21. Est-il possible de diminuer le nombre de calculs nécessaires pour résoudre le problème précédent ?

Il est possible de ne faire qu'une double boucle imbriquée en testant si $R^2-(x-a)^2-(y-b)^2$ est le carré d'un entier : la complexité de l'algorithme serait alors de $\mathcal{O}(n^2)$